HawkFlow: Scheme for Scalable Hierarchically Distributed Control in Software Defined Network

Xiangyang Zhu, Bing Chen, and Hongyan Qian
Institute of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Jiangsu, 211106, China
Email: {zxycly, cb_china, qhy98}@nuaa.edu.cn

Abstract —Compared with the traditional design of network architecture, Software Defined Network (SDN) can be programmed to provide more flexible, fine-grained and differentiated services because of its control centralization property. However, with a variety of network functions such as firewall and multicast are gradually added to the SDN controller, the heavy computational load on SDN control plane has made it the bottleneck of whole network architecture in large scale networks. Among all the solutions proposed in the literature, distributed control plane is very promising in solving the problem. This paper presents HawkFlow, a scheme based on hierarchically distributed control plane, to improve the efficiency and scalability of SDN control plane. HawkFlow proposes blocking island theory and network aggregation mechanism to reduce the searching space of Centralized Single Controller Routing (CSR) algorithms. Routing requests are divided into three levels according to the destination IP address, in which routing processes in local networks are designed to be CSR to reduce the average routing complexity. Experimental results show that the mechanisms discussed in this paper can greatly improve the efficiency of hierarchically distributed control plane, especially in the networks with large proportions of local network traffic such as data centers or campus networks.

Index Terms—Software defined network, blocking island, network aggregation, distributed routing

I. Introduction

The traditional network architecture lacks of global network view, which makes it unable to dynamically allocate network resources and provide fine-grained, differentiated services. Besides, since network functions are distributed in network devices and "hop-by-hop" routing protocols are adopted, the end-to-end Quality of Service (QoS) cannot be guaranteed. To alleviate these problems, researches on the next generation network have been launched throughout the world. Particularly, Software Defined Network (SDN) has succeeded in drawing the attention of the industry and the academia [1]. SDN decouples the control plane with the forwarding plane, and the most significant feature of SDN is to provide centralized control and global view of the network. SDN abstracts the underlying network resources and proposes to utilize a logically integrated controller to deploy high-level policy. Each switch in the network communicates with the control plane via a secure channel implemented by OpenFlow protocol [2], the most famous and well known technical implementation of SDN. Network administrators and application developers are able to configure, manage and optimize network resources through programming high quality applications.

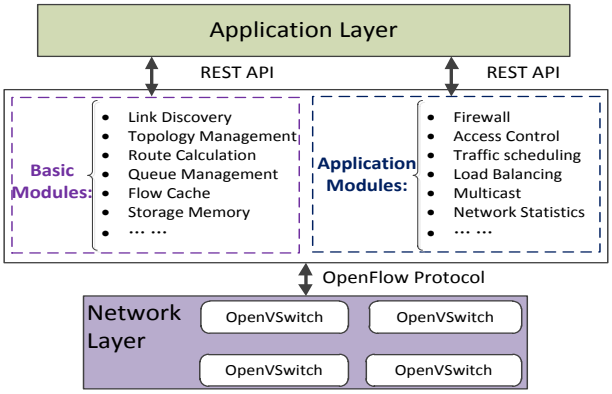


Fig. 1. Control plane: waist of SDN architecture

However, along with advantages, SDN also yields some severe challenges. As is shown in Fig. 1, SDN control plane serves as the brain and "waist" of the whole architecture, which contains many basic function modules, and the characteristics of SDN architecture makes that various network functions are gradually added to the control plane, including firewall, load balancing, access control, resource scheduling, etc. It is just the property of network control centralization that enables the features of programmability and flexibility, but the heavy computational load on SDN control plane may result in scalability problems. On the one hand, the control plane needs to interact with all the switches in the network, while researches show that the amount of data in OpenFlow channel increases linearly with the number of service flows and the size of network [3], on the other hand, centralized control architecture have limited processing capability. According to statistics, NOX [4] controller can only handle 30K path calculation requests per second, while a cluster of 1500 servers can generate 100K new flow requests and a data center composed of 100 switches can generate 10000K new routing computation requests per second. Simulations on the data set "The CAIDA Anonymized 2011 Internet Traces

Manuscript received May 31, 2016; revised October 19, 2016.
Corresponding author email: zxycly@nuaa.edu.cn.
doi:10.12720//jcm.11.10.910-917

Dataset" using a self-defined centralized controller indicate that when the routing request rate exceeds 26K times per second, the delay will increase dramatically, which cannot meet the needs of the real time services [5].

In order to improve the efficiency of SDN control plane, three types of solutions have been proposed in the literature. NOX, Beacon [6] and SNAC [7] try to improve the hardware and performance of SDN controller or to apply more efficient algorithms, but they only provide limited extra scalability and cannot cope with single point failure of controller. DevoFlow [8] and DIFANE [9] attempt to transfer partial control plane tasks to the forwarding plane, but they require modifications in the flow table structure and the hardware of switches. ASIC [10] and Kandoo [11], etc., apply multiple collaborative controllers to improve the capability of SDN control plane, these distributed solutions are considered to be able to solve various problems encountered in SDN.

Compared with Fully Distributed Control Plane (FDP), Hierarchically Distributed Control Plane (HDP) is more promising in addressing the scalability problem, because each controller in FDP needs to maintain the information of global network, which will consume a lot of storage space and bandwidth of OpenFlow channel, while only upper layer of HDP needs to maintain global network status, which makes it more easier to extend with less extra overhead. The main contributions of this paper are as follows:

1) Blocking island theory and network aggregation methods are applied to reduce the average complexity of Centralized Single Controller Routing (CSR) algorithms.
2) The routing requests are divided into three levels to make full use of advantages of CSR.
3) The fully distributed routing is designed to be an ordered sequence of CSR, which greatly reduce the complexity of distributed routing.

The rest of this paper is organized as follows. Section II reviews the related works. In section III the details of HawkFlow are presented, including using blocking island theory and network aggregation to reduce the searching space of CSR, three levels of routing and fully distributed routing algorithm. Section IV presents the simulation environment and results. Finally, Section V concludes the paper.

II. Related Work

There are several non-standard distributed control plane schemes that have been proposed in the literature, they use different information consistency methods and focus on different problems.

HyperFlow [12] stores the controller status in a distributed file system named WheelFS, the network is divided into a number of regions which are managed by local controllers, but all controllers have to maintain the global network information, the fully distributed architecture will reduce the speed of information synchronization between controllers. Onix [13] maintains the consistency of the global network state through Network Information Library (NIB), and it provides a set of programming APIs for customizing flexible network applications. NIB is designed to adopt mature distributed control system solutions, so it may be faced with problems like poor performance and network state inconsistency. Master/Slaves [14] is proposed to mainly improve the reliability of distributed control plane, working controllers are called Masters, backup controllers are called Slaves, when the tasks of Masters will automatically switched to Slaves when they are detected to be failed. Masters are detected to be failed, but the architecture does not concern the scalability problems of distributed control plane. ASIC is designed to solve load balancing problem between controllers, especially when a large number of routing requests arrive at the distributed control plane, ASIC use MySql database to store network information and achieve information consistency, but problems of scalability in large scale networks and distributed routing algorithms are not discussed.

Kandoo is a kind of hierarchical distributed control plane, which puts all frequent operations on the local controllers. The central controller of Kandoo is responsible for the maintenance of the global network status and interactions with applications. Kandoo aims to reduce the information exchange between forwarding plane and central controller, while lots of operations such as all routing requests still need to invoke central controller, it points out that the central controller can also be distributed control plane, but no detailed implementation is proposed. ElastiCon [15] is proposed to adaptively change the number of controllers based on traffic conditions and the load to improve responsiveness of the control plane, but the architecture mainly focus on how to obtain the least number of controllers needed according to the network status, rather than how to obtain best performance of given controllers by designing high efficient distributed control plane architectures.

Among other distributed control plane architectures, Zebra [16] processes routing requests concurrently through a way of shared data view. Kotronis *et al.* [17] propose a control plane model focusing on evolving inter-domain routing so that the legacy BGP remains compatible. Also, Raghavan *et al.* [18] introduce a Software Defined Internet Architecture (SDIA) considering both intra and inter-domain forwarding tasks.

III. The HawkFlow Scheme

To address the scalability and performance issues in SDN, this paper proposes a novel scheme called HawkFlow that leverages several mechanisms to reduce the average complexity of routing algorithms. HawkFlow is based on HDP, for the convenience of discussion, we assume that the upper-layer control plane is fully distributed, and each upper-layer controller (called UC)

administrates several lower-layer controllers (called LC). Each LC manage a local network (called LA) which can be heterogeneous, the network managed by a UC is called UA.

The high complexity of distributed routing may lead to performance decline, but to the best of our knowledge, the advantages of CSR have not been fully utilized in distributed control planes. To reduce the average complexity of routing, we will divide the routing requests into three levels, in which level-1 and level-2 routing are regarded as CSR, and apply blocking island theory and network aggregation mechanism to reduce the searching space of CSR. Aggregated network is computed by LCs and reported to UCs when the system is initiated, when a switch is added, removed or other topology changes take place, a little aggregated information which is related to the changed parts needs to be recomputed and uploaded. The distributed optimal routing is based on CSR, though there are often many constraints and optimization objectives in QoS flows, we will propose a general centralized routing mechanism based on blocking island theory, since multiple objective optimization is not the focus of this paper.

A. Centralized Single Controller Routing (CSR)

We propose centralized routing as a Constraint Shortest Path (CSP) problem. Blocking Island (BI) [19] theory is used to reduce the searching space for CSR algorithms. The key idea of BI is transforming the original network into clusters which contain available resources information. For a CSP problem, it is crucial to find out the QoS constraints and the optimization objects. For example, the typical QoS constraints are bandwidth, packet loss, delay, delay variation, etc., and the optimization objects can be least hops, lowest energy costs, load balancing or other concerned parameters. For simplicity, we take bandwidth as the only QoS constraint and hop-count the only optimization object into account, but note that the conclusions can be generalized to other QoS requirements or cost metrics.

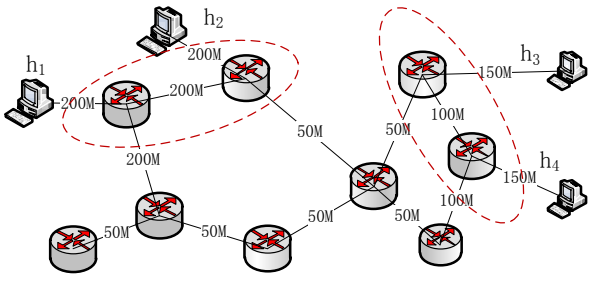


Fig. 2. BI clusters of network

The network we present is denoted as a directed graph $G(N,A)$, where N is the set of nodes and A is the set of links. A routing request is defined by a three tuple $d=(S,T,C)$, where S and T represent the source and destination respectively, and C represents the bandwidth required, $R(S,T)$ represents the set of all routes from S to T. The main idea of BI is to abstract the available network resources (e.g., bandwidth) into a hierarchy tree, the C-BI for a node S is the set of all nodes in the network that can be reached from S with constraint C. The C-BI graph of a network is a graph which divides the network into clusters according to the available bandwidth. Four key properties of BI used in this paper are listed as follows:

1) Route existence: For a given request $d=(S,T,C)$, there are routes existing if and only if S and T are in the same C-BI.
2) Uniqueness: There is only one C-BI for a given node S.
3) Inclusion: If $C_m<C_n$, C_n-BI is the subset of C_m-BI for a given node.
4) Partition: The whole network is partitioned into clusters.

Assuming that the C_1-BI graph of node S_1 has been constructed, the graph contains N nodes, numbered $S_1,S_2,\ldots,S_N$. For a routing request $d=(S_1,S_n,B_{min})$, where $n=1,2,...,N$ and $B_{min}<C_1$, then the algorithm will return "route exist", because S_1 and S_n are in the same C_1-BI. For a routing request $d=(S_1,T,C_1)$ where T is not in C_1-BI, the algorithm will return “route does not exist”, and for $d=(S_1,S_n,B_{min})$ where $n=1,2,...,N$ but $B_{min}>C_1$, then the algorithm will try to construct C_2-BI where $C_2>B_{min}$. A blocking island cluster graph is shown in Fig. 2. If given a request $d=(h_1,h_3,100)$, we immediately know that the route does not exist, because h_1 and h_3 are not in the same 100M-BI; if given $d=(h_1,h_2,150)$, the algorithm will return “route exist” immediately since h_1 and h_2 are in the same 150M-BI. Assume that we set the cost metric as the least hops on the route, then we can use CSP algorithms to compute the optimal route on the 150M-BI of h_1, the nodes not in 150M-BI of h_1 will not be taken into consideration, obviously in this way, the searching space of routing algorithms has been greatly reduced with bandwidth guarantee.

According to the definition of BI, BI graph of a network can be constructed using greedy algorithm shown in Algorithm 1. The average time complexity of constructing and updating BI graph of a node is $O(A^2)$, while the complexity of judging route existence problem for a given request is only $O(1)$. After the BI graph is constructed, CSP algorithms will be executed on it, for a route r, we define the cost as

$$C_r = \sum_{(i,j)\in r} C_{(i,j)} \tag{1}$$

and the bandwidth as

$$B_r = \arg\min_{(i,j)} \{B_{(i,j)} \mid (i,j) \in r\} \tag{2}$$

where $C_{(i,j)}$ is the cost and $B_{(i,j)}$ the bandwidth of link (i, j).

Obviously, the CSP problem can be formulated as

$$r = \arg\min_{r} \{C_r \mid r \in R_{(S,T)}, B_r \geq B_{min}\} \tag{3}$$

Yet we know that the CSP shown in (3) is NP-complete and heuristic algorithm is needed, we propose to use Lagrangian Relaxation Based Aggregated Cost (LARAC) because it can find a good route within average time complexity $O((A+N\log_2^V)^2)$. Since we only consider bandwidth as example, the solution of (3) can be represented as

$$r = CSR(G_{(N,A)},(S,T),B_{min}) \tag{4}$$

In the following we will apply (4) to represent the optimal route of centralized routing.

ALGORITHM 1:BI GRAPH CONSTRUCTION ALGORITHM

```
ConstructBIGraph(N, A, C)
1.V := {∅}
2.for all v in N do
3.  if notVisited(v) then
4.    I := ConstructBIByTraverse(N, A, C, v)
5.    V := L.add(I)
6.  end if
7.end for
8.return V
ConstructBIByTraverse(N, A, C, v)
9.I := {v}
10.S := {links|incident to v and weight ≥ C}
11.while S ≠ {∅} do
12.  e := pop(S)
13.  p := {point|end of e}
14.  if p ∉ I and weight(e) ≥ C then
15.    I := I ∪ {p}
16.    S := S ∪ {links|incident to p and weight ≥ C}
17.  end if
18.end while
19.return I
```

B. Network Aggregation in HawkFlow

The complexity of routing calculation and the communication overhead are proportional to the network size, which is determined by the number of switches in the network, so network aggregation of cost metric and QoS parameters is essential. To construct aggregated network topology which only contains border switches, and each virtual link between border switches represents all physical links, virtual link parameters need to be generated from physical parameters. This paper considers only cost and bandwidth as aggregated parameters, but the method used is also suitable for other QoS parameters.

In order to improve the accuracy of topology aggregation, we introduce the concept of Representing Node (RN) to eliminate partial physical routes. Assuming that $R(S,T)$ represents the set of routes from S to T, for any route r_i in $R(S,T)$, if there exists a route r_j in $R(S,T)$ satisfying that $C_{r_i} \geq C_{r_j}$ and $B_{r_i} \leq B_{r_j}$, then r_i should be eliminated, because its parameters cannot provide higher guaranteed QoS service, the routes remained are denoted as RN. If we map the parameters of all RNs on a rectangular coordinate system, in which the horizontal axis represents the cost and the vertical axis represents the bandwidth, then the corresponding ladder diagram divide the axis area into two parts, one of the parts contains the optimal parameter for the virtual link. In this paper, we propose the fitting algorithm for ladder diagram parameters based on RNs to compress the number of effective RNs and reduce the distortion of network aggregation.

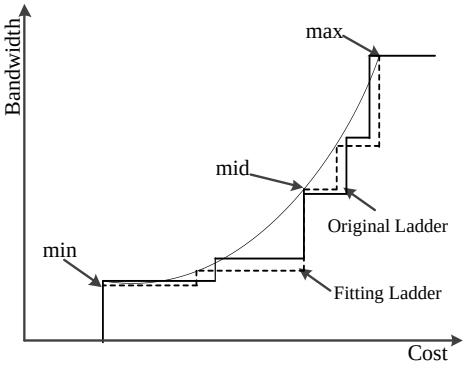


Fig. 3. Parabolic ladder fitting

Since the knee points of RNs often cannot form a straight line, and straight line fitting algorithm may cause large distortion, so we use parabolic based ladder fitting algorithm to solve the problem, the rectangular coordinate system for parabolic ladder fitting is shown in Fig. 3. Assume that the original staircase is denoted as Q, we first generate a parabolic f_q approving all knee points using the least square method, and then f_q is used to generate the fitting ladder diagram S_q. The details are shown in the following steps:

1) The least square method is used to generate the parabolic f_q, and three points $P^q_{min}(C^q_{min},B^q_{min})$, $P^q_{mid}(C^q_{mid},B^q_{mid})$, $P^q_{max}(C^q_{max},B^q_{max})$ are used to represent f_q,

$$\begin{cases} C^q_{min} = C^s_{min} \Rightarrow B^q_{min} = f_q(C^q_{min}) \\ C^q_{mid} = C^s_{mid} \Rightarrow B^q_{mid} = f_q(C^q_{mid}) \\ C^q_{max} = C^s_{max} \Rightarrow B^q_{max} = f_q(C^q_{max}) \end{cases} \tag{5}$$

in which C^s_{min} , C^s_{mid} and C^s_{max} represent the RN points with least cost, middle cost and largest cost respectively.

2) Two intermediate parameters are defined,

$$\begin{cases} b_1 = (B^q_{mid} - B^q_{min})/(mid-1) \\ b_2 = (B^q_{max} - B^q_{mid})/(m-mid-1) \end{cases} \tag{6}$$

in which min, mid and max represent the least point, the middle point and the max point of parabolic f_q respectively, and m represents the number of RN points.

3) The cost of RNs on fitting ladder diagram S_q can be updated as

$$C_i^q = \begin{cases} C_{min}^q + (i-1)\times b_1, 1 < i < mid \\ C_{mid}^q + (i-mid-1)\times b_2, mid < i < max \end{cases} \quad (7)$$

4) The bandwidth of RNs can be updated using f_q,

$$B_i^q = f_q(C_i^q), 1 \le i \le max \quad (8)$$

The fitting ladder can be calculated using (5), (6), (7) and (8), and the ladder can be represented using seven key fields $(C_{min}^q, B_{min}^q, C_{mid}^q, B_{mid}^q, C_{max}^q, B_{max}^q, m)$. In this way, the parameters of the virtual link can be represented simply, so the storage of physical routes can be compressed effectively.

After generating the fitting ladder diagram S_q, we first calculate K-disjoint minimum cost routes $r_1,r_2,\ldots,r_K$ from S_q using Dijkstra algorithm, when calculating route r_k (k=1,2,…,K) , links in route r_1 to route r_{k-1} are removed from the network, then the average cost and QoS parameters of routes $r_1,r_2,\ldots,r_K$ will be assigned to the virtual link (S,T). Take bandwidth for example, suppose that the cost of virtual link is $C_{(S,T)}$ and the bandwidth of virtual link is $B_{(S,T)}$, (1) and (2) can be used to calculate the cost C_r and the bandwidth B_r of route r, then the cost of virtual link (S,T) is

$$C_{(S,T)} = \frac{1}{K}\sum_{i=1}^{K} C_{r_i} \quad (9)$$

The bandwidth of virtual link (S,T) is

$$B_{(S,T)} = \frac{1}{K}\sum_{i=1}^{K} B_{r_i} \quad (10)$$

It is worth noting that in the method we determine the cost and bandwidth of the virtual link mentioned above, we have to calculate K-disjoint routes, which may unable to reach the ideal efficiency. Actually, we also can simply use the parameters of minimum cost route or maximum cost route as the parameters of virtual link, which method to be adopted should be decided according to the specific circumstances. The essence of topology aggregation of LA is to transfer part of upper-layer control plane computation tasks to the lower-layer control plane.

C. Three Levels of Routing in HawkFlow

The performance of distributed control plane cannot increase linearly with the number of controllers, one of reasons is that information transmission and network status synchronization between controllers will bring extra overhead, but a more important reason is that the complexity of distributed routing is much higher than CSR. As we know, there exists a large amount of local traffic in networks like data centers, these local traffic routing requests should not be treated as distributed routing generally. To make full use of the advantage of CSR, the routing requests in HawkFlow are divided into three levels. In addition, this paper proposes a distributed routing algorithm based on CSR. Routing requests in HawkFlow can be divided into three levels because:

1) Through judging the header fields such as IP address and VLAN id, OpenFlow is capable of differentiating service flows into three types or network levels, then "divide and conquer" strategy can be used to handle these different types of traffic.
2) The CSR algorithm discussed in Section III-A is suitable for local routing in a LA or an aggregated UA, because the network size of them is relatively small. On the contrary, inter-UA routing may involves a large number of switches, so distributed routing with two or more UAs , which is more complicated than CSR is needed.

Notations will be used in later discussions are listed in Table I.

TABLE I: NOTATIONS TO BE USED

Notation	Description
G_g	Global network
G_a	Aggregated global network
G_{li}	LA, where i=1,2,…,L, L is the number of LAs
G_{ali}	Aggregated LA
G_{ui}	UA, where i=1,2,…,U, U is the number of UAs
G_{aui}	Aggregated UA
G_{agi}	Aggregated global network, except UA_i and destnation LA, where UA_i represents source UA in the ith step of global distributed routing
S_{bs}	Set of border switches

1) Intra-LA routing (level-1 routing)

When a routing request packet is reported to a LC, the header fields will be analyzed and the source IP and destination IP will be extracted to determine the network level of the service flow. If the source IP and destination IP are judged to be in the same LA, then LC will initiate a process of intra-LA routing, otherwise the packet will be forwarded to its UC for further analysis. As mentioned above, we take intra-LA routing as CSR problem to take advantage of the low complexity of centralized routing algorithms. Assuming that the source switch is S, the destination switch is T, and the bandwidth required is B_{min}, then intra-LA routing can be formulated as $r=CSR(G_{li},(S,T),B_{min})$, where i=1,2,…,L, L is the number of LAs, and S and T are in LA_i, the expression is the centralized routing discussed in Section III-A.

2) Intra-UA routing (level-2 routing)

Similarly, when a UC receives a routing request packet from one of its LCs, it will judge whether the destination node T is in its UA, if S and T are in the same UA, then UC will initiate a process of intra-UA routing, otherwise the packet will be published to other UCs. Intra-UA routing is also designed to be CSR problem, the difference is that intra-UA routing involves the aggregated UA topology and UC also participates in global distributed routing. Each LC uploads both original topology and aggregated topology to its UC, and then UC can calculate the aggregated UA topology by combining these topologies. The optimal path of intra-UA routing can be given as $r=CSR(G_{aui},(S,T),B_{min})$, where i=1,2,...,U, U is the number of UAs, the source S and the destination

T are in the same UA but not in the same LA. The virtual links in the routing will be replaced by LCs with real links stored when the aggregated versions of LAs are obtained.

3) Global fully distributed routing (level-3 routing)

Intra-LA and intra-UA routing are designed to be CSR, which can reduce the average routing complexity. When UC receives a routing request packet and determines that the destination *T* is not in its UA, it will launch a process of global fully distributed routing. Global distributed routing is designed as an ordered list of CSR based on aggregated global topology from source *S* to destination *T*. Assuming that it needs *I* steps of CSR for a particular routing request $d=(S,T,B_{min})$, S_i is the source node of step *i* ($S_1=S$), and UA_i is the UA where S_i lies, each step *i* (i=1,2,…,I) will determine four elements of the final route: the links inside UA_i, the outgoing node of UA_i, the incoming node of UA_{i+1} and the inter-UA route between UA_i and UA_{i+1}. The last step only determines the final route inside UA_i, and the incoming node of UA_{i+1} is also the source node of step i+1.

The optimal route of step *i* can be formulated as, $r_i=CSR(G_{agi},(S_i,T),B_{min})$, where i=1,2,…,I, G_{agi} is the global aggregated network except UA_i and the destination LA, S_i is the source node in UA_i of step *i*, *T* is the destination node and B_{min} is the required bandwidth. For step *i*, the route inside UA_i is

$$r_{UA_i} = r_i \cap UA_i \tag{11}$$

The outgoing node of UA_i is

$$Out_{UA_i} = r_{UA_i} \cap S_{bs} \tag{12}$$

The source node of Step i+1 is

$$S_{i+1} = \{In_{UA_i} \mid (Out_{UA_i}, In_{UA_i}) \in r_i\} \tag{13}$$

The link between UA_i and UA_{i+1} is

$$Link_i = (Out_{UA_i}, S_{i+1}) \tag{14}$$

and the distributed optimal route can be formulated as

$$\begin{aligned} r &= \bigcup_{i=1}^{I} (Link_i \cup r_{IA_i}) \\ &= \bigcup_{i=1}^{I} (r_i \cap UA_i \cap S_{bs}, S_{i+1}) \cup (r_i \cap UA_i) \end{aligned} \tag{15}$$

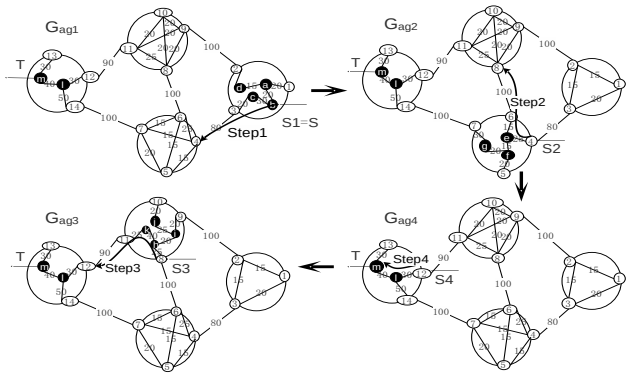


Fig. 4. Example of global distributed routing

Take the simple topology in Fig. 4 as an example, assume that the routing request is $d=(S,T,20)$, it will take 4 steps of CSR in total for the fully distributed routing, in which step 1 is based on G_{ag1} with UC_1 and $r_1=((b,c,3),(3,4),4)$, step 2 compute CSR based on G_{ag2} with UC_2 and $r_2=((4,e,6),(6,8),8)$, step 3 compute CSR based on G_{ag3} with UC_3 and $r_3=((8,h,k,11),(11,12),12)$, and step 4 is based on G_{ag4} with UC_4, $r_4=(12,l,m)$. Then the final distributed optimal route can be obtained easily by combining the results of these steps.

IV. Evaluation

In this section, we will test the efficiency of the scheme proposed through simulations. The experiment topology is emulated by Mininet 2.2.1 [20] and connected to a remote control plane implemented using Floodlight 1.1.0 [21]. The major network topology used is consisted of 7 LAs and 3 UAs, in which LA_1 to LA_2 belong to UA_1, LA_3 to LA_5 belong to UA_2 and LA_6 to LA_7 belong to UA_3, each LA includes 20 switches. Border switches are selected randomly, the bandwidth of intra-LA links are set to be 400Mbps and other links 800Mbps. Mausezahn 0.38.1[22], an open-source network traffic generator, is used to generate data flows which can be customized freely. The whole network is designed using GI-ITM tools [23], and then Python API provided by Mininet is used to generate the network.

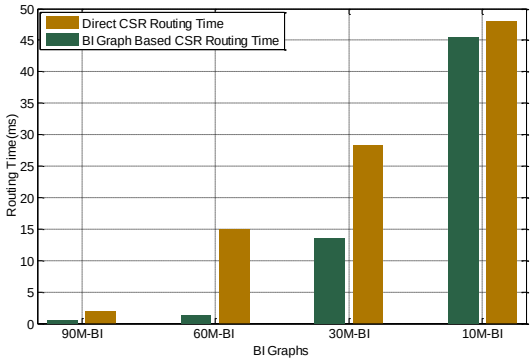


Fig. 5. Latency of direct and BI based CSR routing

We first test the delay under different network loads and different proportions of local network traffic. Four different proportions of local network traffic are selected for comparison: a) all routing requests are regarded as level-3 routing, b) 10% level-1 routing and 10% level-2 routing, c) 20% level-1 routing and 20% level-2 routing, d) 30% level-1 routing and 30% level-2 routing. Experiment result show that the average delay increases with the routing request rate, but level-1 and level-2 routing make a great difference to the average delay of HawkFlow. Generally speaking, if both level-1 and level-2 routing account for 30% of total routing requests, the average delay will be reduced by 40% compared to the case with no local network traffic. The experiment result shows that dividing routing requests into three levels can indeed improve the processing efficiency and capability of the control plane.

In the discussion of CSR routing, BI is used to reduce the searching space for CSP algorithm, especially in a large network. To compare the delay of direct CSR routing with BI graph based CSR routing proposed in this paper, a network with 1K switches is constructed for extensive experiments. The bandwidth of each link is randomly allocated from 50M to 100M. Four types of routing requests are designed for 90M-BI, 60M-BI, 30M-BI and 5-BI respectively. The comparison results are shown in Fig. 5. The BI graphs used in the simulation can be constructed in about 2 seconds, they are calculated when the whole system is initiated, BI graph based routing have higher efficiency in most cases because they can shrinks the searching space significantly, except for the 5M-BI graph whose network size is very close to the whole graph. Since constructing BI graphs may need a short time, this may bring a little influence on the CSR computing time in the initial stage of the system, but this influence can be ignored with the increase of routing requests.

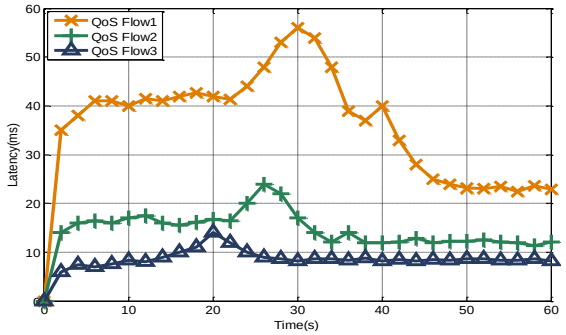


Fig. 6. Latency before/after network aggregation

In order to estimate the influence of network aggregation, three QoS flows are specified: a) QoS flow1 is a cross-UA traffic from LA1 to LA7 with bandwidth 100Mbps required, b) QoS flow2 is a cross-LA traffic from LA3 to LA5, the bandwidth required is 50Mbps, c) QoS flow3 is a intra-LA flow in LA1, the bandwidth required is 50Mbps. From Fig. 6, it is easy to note that before network aggregation is enabled through REST API (0~15s), the average routing latency is about 20ms, and the latency of QoS flow1 is much higher than QoS flow2 and QoS flow3. After 15s, when network aggregation is enabled, the average route computing latency falls to about 14ms. Although QoS flow3 is not benefit from network aggregation, because its routing requests belong to level-1 routing which is based on the original topology, but the latency of QoS flow1 and QoS flow2, whose routing requests belong to level-2 and level-3, is reduced dramatically. Notice that the latency of three flows may rise after network aggregation is enabled, and QoS flow3 is affected prior to other two QoS flows, because the computing of aggregated topology and QoS parameters will consume part of CPU and the bandwidth of OpenFlow channel, and LCs should first calculate the aggregated LA, which then be combined by UCs to form aggregated UA and global aggregated network.

In Fig. 7, we compare HawkFlow with classical HDP and Kandoo on average routing latency under different routing request rate. The data flows in network are randomly generated without customizing the proportion of local network traffic to simulate the real network environment. When routing request rate is above 60K/s, the average latency of HDP shows a sharp rise, and when routing request rate is above 80K/s, the performance of Kandoo begin to decline quickly, while HawkFlow extends the number to 110K/s, and it shows a slowly rising trend at 120K/s.

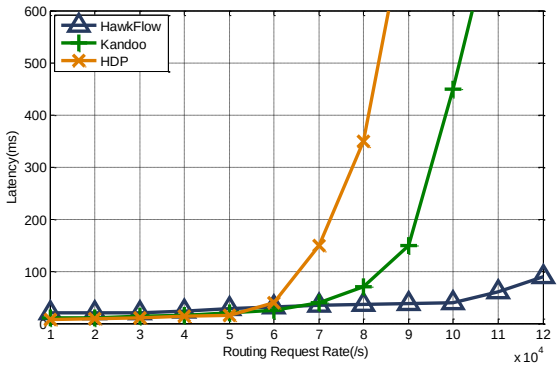


Fig. 7. Performance comparison of three schemes

V. Conclusion

This paper presents a scheme called HawkFlow to improve the efficiency of hierarchically distributed control plane. Blocking island theory and network aggregation are used to reduce the searching space of routing algorithms, and routing requests are divided into three levels to reduce the average time complexity of routing algorithms. Experiment results show that HawkFlow scales remarkably than Kandoo and HDP, especially when local network traffic accounts for a large proportion, such as in data centers and campus networks, we also verify the efficiency of CSR based on BI in large scale networks.

In addition to Floodlight, we plan to apply other open-source SDN controllers such as NOX for more experiments, and we will add back-up controllers to prevent component failures and research for fast-recovery techniques.

References

[1] S. Gorlatch, T. Humernbrum, and F. Glinka, "Improving QoS in real-time internet applications: From best-effort to software-defined networks," in *Proc. IEEE International Conference on Computing, Networking & Communications*, 2014, pp. 189-193.

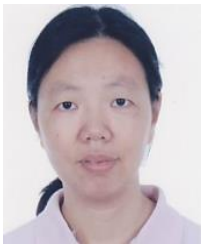
[2] A. Al-Najjar, S. Layeghy, and M. Portmann, "Pushing SDN to the end-host, network load balancing using OpenFlow," in *Proc. IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2016, pp. 1-6.

[3] A. V. Akella and K. Xiong, "Quality of Service (QoS)-guaranteed network resource allocation via Software

Defined Networking (SDN)," in *Proc. IEEE International Conference on Dependable*, 2014, pp. 7-13.
[4] N. Gude, T. Koponen, J. Pettit, B. Pfa, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards and operating system for networks," in *Proc. ACM SIGCOMM Computer Communication Review*, Jul. 2008, vol. 38, pp. 105-111.
[5] K. Claffy, D. Andersen, and P. Hick, "The caida anonymized 2011 internet traces equinix-chicago," Mar. 2011.
[6] D. Erickson, "The beacon openflow controller," in *Proc. 2nd ACM SIGCOMM* Workshop *on Hot Topics in Software Defined Networking*, 2013, pp. 13-18.
[7] SNAC: Simple Network Access Control. [Online]. Available: http://www.openflow.org/wp/snac/
[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, and P. Yalagandula, "Devoflow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Computer Communication Review*, 2015, pp. 254-265.
[9] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proc. ACM SIGCOMM Computer Communication Review*, 2010.
[10] P. Lin, J. Bi, and H. Hu, "Asic: An architecture for scalable intra-domain control in openflow," in *Proc. 7th IEEE International Conference on Future Internet Technologies*, 2012, pp. 21-26.
[11] J. Zhou, D. Cheng, W. Wang, R. Jin, and X. Wu, "The deployment of routing protocols in distributed control plane of SDN," *IEEE Trans. Scientific World Journal*, pp. 19–24, 2014.
[12] A. Tootoonchian and Y. Ganjali, "A distributed control plane for openflow," in *Proc. INM/WREN*, 2010, p. 3.
[13] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, 2010, pp. 351-364.
[14] E. S. Spalla, D. R. Mafioletti, A. B. Liberato, C. Rothenberg, L. Camargos, *et al.*, "Resilient strategies to SDN: An approach focused on actively replicated controllers," in *Proc. 33th Brazilian Symposium on Computer Networks and Distributed Systems*, 2015, pp. 246-259.
[15] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Software Defined Network*, 2013, pp. 7-12.
[16] H. Yu, K. Li, H. Qi, W. Li, and X. Tao, "Zebra: An east-west control framework for SDN controllers," in *Proc. IEEE International Conference on Parallel Processing*, 2015, pp. 610-618.
[17] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: Better internet routing based on sdn principles," in *Proc. 11th ACM Workshop Hot Topics Network*, 2012, pp. 55-60.
[18] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software defined internet architecture: Decoupling architecture from infrastructure," in *Proc. 11th ACM Workshop Hot Topics Network*, 2012, pp. 43-48.
[19] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "CheetahFlow: Towards low latency software-defined network," in *Proc. IEEE International Conference on Communications*, 2014, pp. 3076-3081.
[20] Mininet: Open Source Network Emulator. [Online]. Available: http://mininet.org/
[21] Floodlight: Component-based SDN Framework. [Online]. Available: http://www.projectfloodlight.org/floodlight/
[22] Mausezahn: Fast traffic generator. [Online]. Available: http://www.perihel.at/sec/mz/mzguide.html/
[23] GT-ITM: Georgia Tech Internetwork Topology Mod. [Online]. Available: http://www.cc.gatech.edu/gtitm/.

Xiangyang Zhu was born in Jiangsu Province, China, in 1987. He received the B.S. degree in Automatic from Nanjing University of Posts and Telecommunications, Jiangsu Province, China, in 2010. He is currently working toward Master's degree at Nanjing University of Aeronautics and Astronautics, Jiangsu Province, China. His research interests include computer networks and Software-defined networking.

Bing Chen, born in 1970, earned B.S. and M.S. degree from the department of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, Jiangsu Province, China, in 1992 and 1995, respectively. He earned Ph.D. degree in the College of Information Science and Technology at NUAA in 2008. Now he is a professor in the College of Information Science and Technology at NUAA. His main research interests are computer network and embedded system.



Hongyan Qian was born in 1973. She received the B.S. and M.S. degrees in computer engineering and Ph.D. degree in information science and technology from the Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, Jiangsu Province, China, in 1995, 1998, and 2010, respectively. Her main research interests are computer networks and wireless communications.